

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
 Кафедра Информационных систем и технологий
 Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
 Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА к курсовому проекту на тему:

Web-приложение «Планировщик бюджета»

Выполнил студент Мацуев Илья Михайлович
(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Дубовик М.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Дубовик М.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2020

Реферат

Пояснительная записка курсового проекта состоит из 48 страниц, 43 рисунка, 3 приложений, 6 источников литературы.

Основная цель курсового проекта: разработка web-приложения «Планировщик бюджета».

В первом разделе рассматриваются основные технологии, которые использовались в разработке данного приложения, постановка задач, а также обзор аналогов.

Во втором разделе описана архитектура курсового проекта.

В третьем разделе предоставлена информация о разработке приложения, объектах базы данных.

Четвертый раздел содержит описание процесса запуска курсового проекта в docker контейнер.

Пятый раздел содержит руководство пользователя для разработанного клиентского приложения.

В шестом разделе представлены результаты тестирования приложения.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

Содержание

Реферат	2
Введение	4
1 Постановка задачи	5
1.1 Алгоритмы решения	5
1.2 Обзор прототипов	6
1.2.1 Money Lover	6
1.2.2 Spendee	7
1.2.3 Wallet	8
2 Проектирование программного средства	10
2.1 Проектирование архитектуры проекта	10
2.2 Разработка диаграммы вариантов использования	13
2.3 Структура моделей и классов	15
2.4 Структура курсового проекта	16
3 Разработка программного средства	19
3.1 Описание функциональных возможностей приложения	19
3.2 Разработка базы данных	19
3.3 Описание алгоритма смены пароля пользователя	24
3.4 Процесс создания транзакций	25
4 Руководство программиста	27
5 Руководство пользователя	28
5.1 Главная страница	28
5.2 Добавление единичной транзакции	28
5.3 Секция оповещений	30
5.4 Страница транзакций	32
5.5 Страница категорий	34
5.6 Страница настроек	35
6 Тестирование	37
Заключение	41
Список литературы	42
ПРИЛОЖЕНИЕ А	43
ПРИЛОЖЕНИЕ Б	44
ПРИЛОЖЕНИЕ В	46

Введение

В данном курсовом проекте разработано WEB-приложение «Планировщик бюджета». Логически оно разделено на две части: серверную, написанную на ASP.NET Core [1], и клиентскую, которая написана с помощью JavaScript-библиотеки React [3] с использованием Apollo Client [2], физически же WEB-приложение представляет собой одно приложение.

Чтобы стать богатым человеком, нужно уделять внимание не только тому, сколько вы зарабатываете, но и тому, сколько тратите. Человек постоянно старается извлечь максимальную выгоду из любой сделки и любого случая, поэтому всегда пользовались спросом приложения и утилиты, позволяющие улучшить уже существующие процессы менеджмента. Один из таких процессов – это подход к мониторингу состояния человека или компании. Реализовать такой подход в настоящее время достаточно просто — необходимо начать отслеживать свои расходы и доходы с помощью одного из специальных приложений для мобильных телефонов или персональных компьютеров.

В сети существует огромное количество подобного рода приложений, однако очень многие из них не подходят для повседневного использования или попросту неудобны. Этот проект призван перенять лучшие качества и нивелировать худшие качества аналогов с целью создания приложения, которое было бы лучше других как со стороны пользователя, так и со стороны бизнеса.

Подводя итоги: целью курсового проекта является разработка WEB-приложения «Планировщик бюджета», которое поможет вам осуществить вашу мечту. Я сделал выбор своей курсовой темы в связи с актуальностью данной задачи в современном мире. Данное приложение содержит тот функционал, который поможет сохранить вам ваши миллионы для лучшей жизни.

1 Постановка задачи

Перед началом разработки необходимо определить цели, задачи, все варианты использования программного средства. Для этого необходимо построить диаграмму использования, которая приведена во второй главе, раздел 2.2.

Постановка задачи – это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются функциональные требования, определяющие функции, которые должно выполнять программное обеспечение, и эксплуатационные требования, определяющие характеристики его функционирования.

Этап постановки задачи заканчивается разработкой технического задания с принятием основных проектных решений.

Главная задача данного приложения – начать контролировать свои расходы и доходы. Проблема нашего поколения заключается в том, что люди не понимают, куда и почему так быстро тратятся деньги. И чтобы решить данную проблему, первое, что необходимо сделать, – это взять её под контроль, в этом вам и поможет разработанное мною веб-приложение «Планировщик бюджета». Данное веб-приложение должно выполнять основные функции по созданию учётной записи пользователя, его личного счёта, возможность добавлять и редактировать расходы и доходы, а также смотреть отчетность. В данном курсовом проекте требовалось реализовать следующие задачи:

- сохранение рабочей информации в централизованной базе данных;
- регистрация одиночных и повторяющихся денежных транзакций;
- возможность получения оповещений об отчетах бюджета за месяц;
- анализ бюджета и транзакций на основе расходов и доходов за промежутки времени;
- настройка параметров работы приложения для пользователя.

1.1 Алгоритмы решения

После окончательной постановки задач необходимо приступить к их решению.

Когда требуется решить задачу, первое, что должно быть сделано, – исследовать ее. Необходимо установить, что дано, что нужно сделать. Надо быть уверенным, что задача решается, иначе последующие шаги могут состоять из напрасных усилий.

После исследования задачи рассматриваются шаги, которые требуются для решения, и порядок, в котором они должны быть выполнены. Шаги, которые необходимы для решения задачи и их последовательность, – это и есть алгоритм.

Когда мы будем удовлетворены тем, как выполняется задание, мы можем

снова вернуться к предыдущим шагам для их улучшения. И так этот цикл может повторяться несколько раз, пока мы не будем удовлетворены полностью. В компьютерной терминологии такой цикл называют циклом разработки программного обеспечения.

Как пример в этой главе можно привести процесс создания и обслуживания повторяющейся транзакции в приложении. Во-первых, приложение накладывает некоторого рода валидации на создание повторяющихся транзакций, одна из таких валидаций это возможность назначать дату транзакций только в будущем времени. После указания заголовка, суммы, категории, даты и интервала повторяющейся транзакции происходит создание новой записи *Regular Transaction* в базе данных. И до тех пор, пока не наступит дата этой транзакции, баланс пользователя не изменится. Реализовано это с помощью библиотеки *Quartz* и специальных классов – “*Jobs*”. Такие классы представляют собой функцию, выполняющуюся и повторяющуюся в строка заданное время. В моем случае работает один из таких воркеров, который запускается каждый день в 2 часа ночи, выбирает все повторяющиеся транзакции из базы данных и создает для каждой из таких транзакций обычную транзакцию для пользователя с соответствующими значениями заголовка, категории и суммы. После этого, время следующей транзакции у повторяющихся транзакций меняется в соответствии с интервалом, указанным для них.

Таким образом пользователь всегда имеет актуальный баланс в соответствии со всеми своими запланированными транзакциями.

1.2 Обзор прототипов

1.2.1 Money Lover

В качестве одного из прототипов было выбрано приложение «*MoneyLover*». *MoneyLover* — настоящем комбайне для всех любителей считать деньги. В отличие от многих других утилит, *MoneyLover* — не просто про учет расходов или планирование бюджета. Программа охватывает практически все возможные сценарии использования денег, а стало быть — большинство ситуаций, которые могут возникнуть на жизненном пути.

Существуют версии этого приложения для всех платформ, в том числе и веб-версия.

Пользовательский интерфейс приложения продемонстрировано на рисунке 1.1.

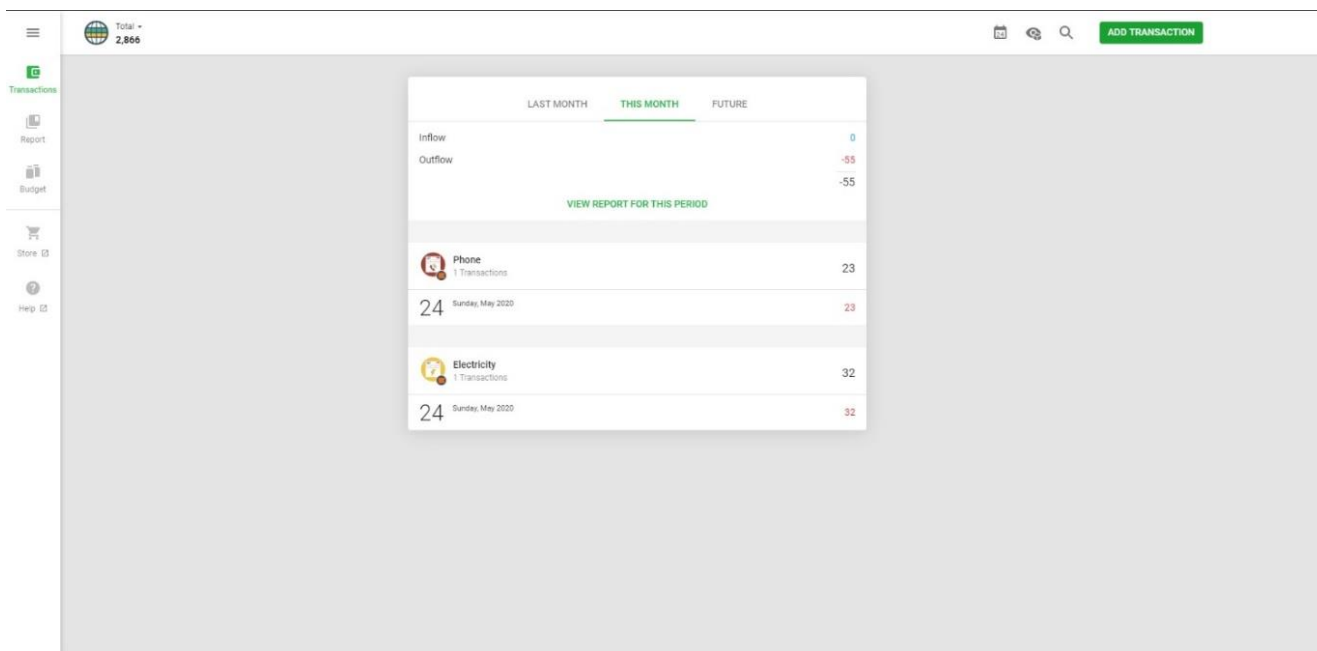


Рисунок 1.1 – Пользовательский интерфейс приложения «MoneyLover»

Приложение предназначено для учёта и анализа собственных расходов и доходов. Приложение имеет довольно красивый и понятный любому пользователю интерфейс и функционал.

Достоинства приложения:

- богатые функциональные возможности;
- продвинутое управление счетами и категориями;
- функция сканирования чеков;
- учет долгов и постановка финансовых целей;
- учет задолженности и доступного лимита по кредитным картам;
- совместный учет финансов;
- синхронизация транзакций с PayPal;
- автоматическая и ручная конвертация валюты;
- информативный виджет.

Недостатки приложения:

- средний уровень безопасности;
- рекламные окна, мешающие работе;
- необходимость регистрации.

1.2.2 Spendee

Spendee выделяется на фоне остальных приложений красивым интерфейсом. Здесь нет унылых таблиц, напоминающих о скучной бухгалтерской работе. Вместо этого Spendee предлагает удобный и

привлекательный UI, чем-то напоминающий ленту в соцсети. Ваши доходы и расходы будут представлены в виде красивой инфографики, так что вы сможете с лёгкостью понять, что происходит с вашими деньгами.

Приложение Spendee существует с версиями под все мобильные устройства, в том числе и веб-версия.

Отличительной особенностью является возможность привязки банковских счетов для автоматического заполнения транзакций по счетам.

Пользовательский интерфейс приложения продемонстрирован на рисунке 1.2 ниже.

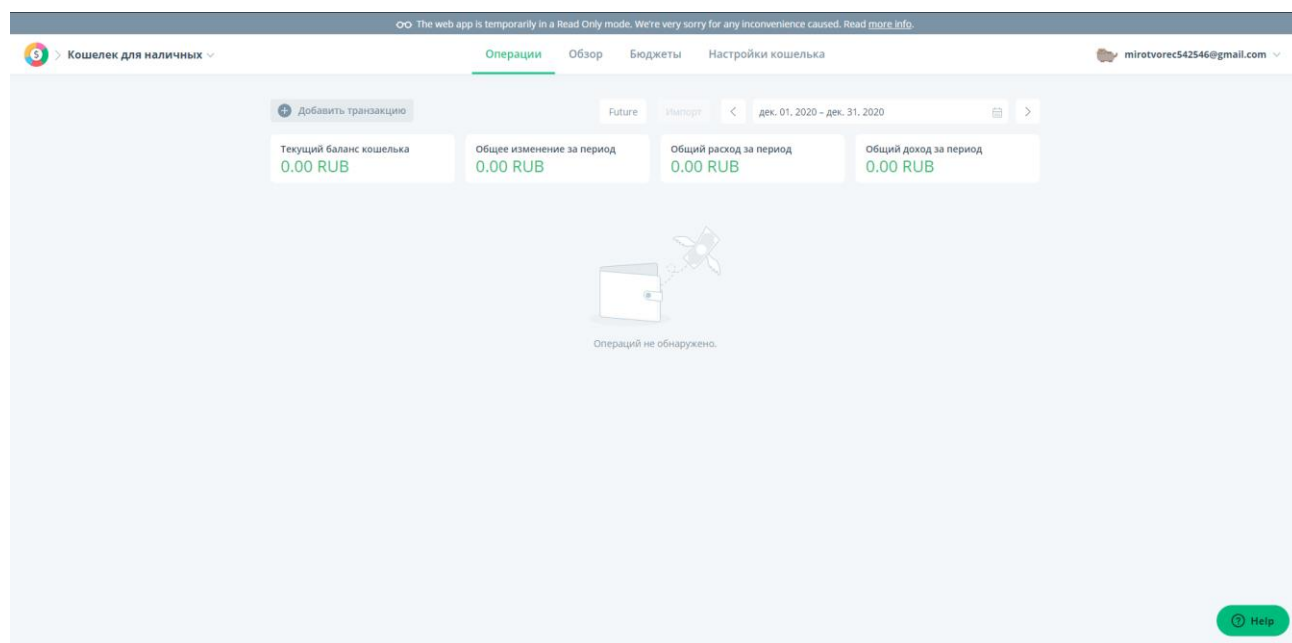


Рисунок 1.2 – Пользовательский интерфейс приложения «Spendee»

Так же приложение имеет возможность создавать кошельки в разной валюте, добавлять транзакции вручную, а так же просматривать статистику за определенный промежуток времени.

Достоинства приложения:

- в приложении можно спланировать расходы по каждой отдельной категории;
- можно вести совместный бюджет с другими аккаунтами.

Недостатки приложения:

- нет возможности привязать к аккаунту больше одного кошелька.

1.2.3 Wallet

Это очень популярное приложение для контроля расходов. Оно поддерживает различные валюты и помогает точно планировать бюджет.

Использовать его можно бесплатно, но если вы оформите платную подписку, то сможете синхронизировать свои банковские транзакции между несколькими устройствами и автоматически классифицировать их, а также добавлять несколько банковских счетов (в бесплатной версии можно использовать не больше трёх).

Пользовательский интерфейс приложения показан на рисунке 1.3.

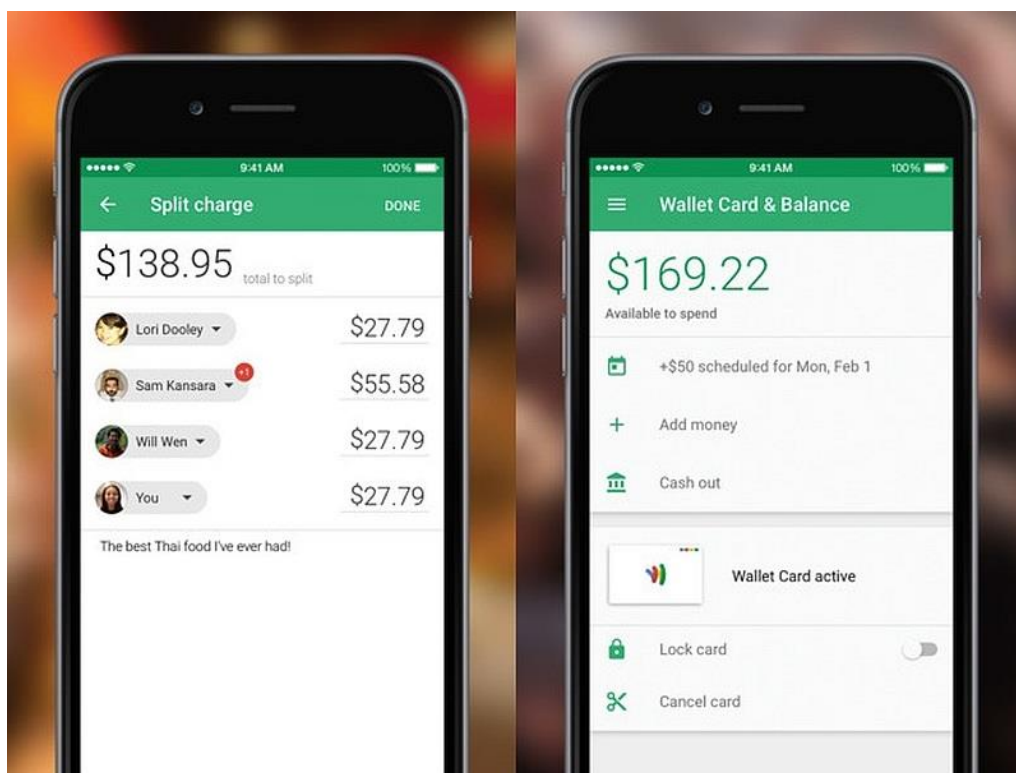


Рисунок 1.3 – Пользовательский интерфейс приложения «Wallet»

Помимо этого, приложение может создавать шаблоны платежей и списки покупок, а также экспортировать данные о ваших доходах и расходах в различные форматы.

Достоинства приложения:

- приложение синхронизируется с банковскими операциями;
- планировать бюджет для определенных покупок помогает функция «создать цель»;
- приложение показывает актуальные курсы валют.

Недостатки приложения:

- нет возможности создавать цели в разных валютах.

2 Проектирование программного средства

2.1 Проектирование архитектуры проекта

Хорошая архитектура – это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать.

То есть, на самом деле можно сформулировать список вполне разумных и универсальных критериев:

1. Масштабируемость (Scalability) – возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей;
2. Ремонтопригодность (Maintainability) – изменение одного модуля не требует изменения других модулей;
3. Заменяемость модулей (Swappability) – модуль легко заменить на другой;
4. Возможность тестирования (Unit Testing) – модуль можно отсоединить от всех остальных и протестировать / починить;
5. Переиспользование (Reusability) – модуль может быть переиспользован в других программах и другом окружении;
6. Сопровождаемость (Maintenance) – разбитую на модули программу легче понимать и сопровождать.

Архитектура – это организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением.

В первую очередь следует, конечно же, стремиться к тому, чтобы модули были предельно автономны. Поэтому проводить ее нужно таким образом, чтобы модули изначально слабо зависели друг от друга.

Для разработки данного проекта была выбрана архитектура клиент-сервер, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями и серверными приложениями.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодействия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

Для разработки данного проекта была выбрана трёхзвенная архитектура, которая реализуется на основе модели сервера приложений, где сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере.

Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае

двухзвенная клиент-серверная архитектура становится трехзвенной (three-tier, 3-tier).

Представление данных – на стороне клиента. Прикладной компонент – на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО). Управление ресурсами – на сервере БД, который и представляет запрашиваемые данные.

Как правило, третьим звеном в трехзвенной архитектуре становится сервер приложений, таким образом компоненты распределяются как показано на рисунке 2.1.



Рисунок 2.1 – Трехзвенная клиент-серверная архитектура

Представление данных – на стороне клиента. Прикладной компонент – на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО). Управление ресурсами – на сервере БД, который и представляет запрашиваемые данные.

В решении курсового проекта для сервера была использована MVC архитектура, которая представлена на рисунке 2.2.

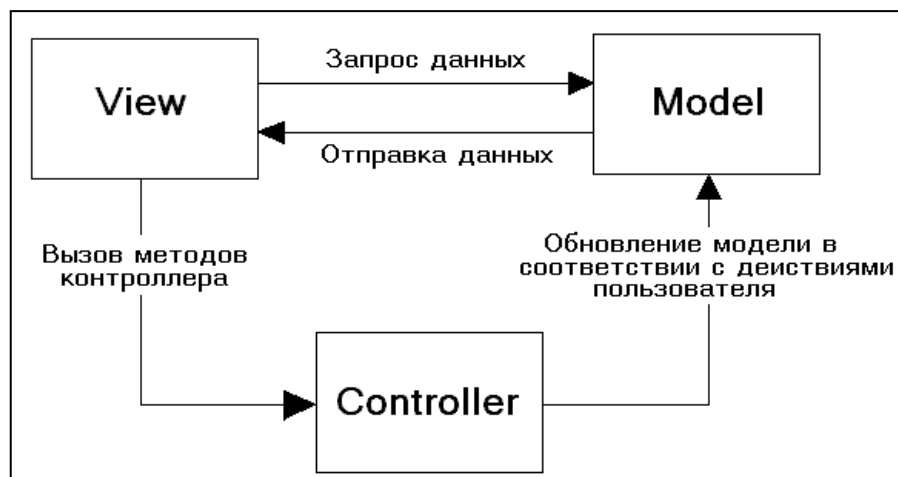


Рисунок 2.2 – Общая схема MVC архитектуры

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента:

Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления. Представление (view) — это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт. Модель (model) представляет класс, описывающий логику используемых данных.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью.

Приложение-клиент представляет собой веб-приложение, написанное на языках JavaScript, HTML, CSS придерживаясь технологии React [3].

React — популярная технология для Frontend-разработки, как и Angular — фреймворк для JavaScript, но это только View (Представление) слой, а значит, у нас в распоряжении только V от MVC — Model-View-Controller [4] (Модель — Представление — Контроллер) архитектуры. React часто упоминается среди других фреймворков, но все, что он дает — это View). Архитектура React приложения показана на следующем рисунке 2.3.

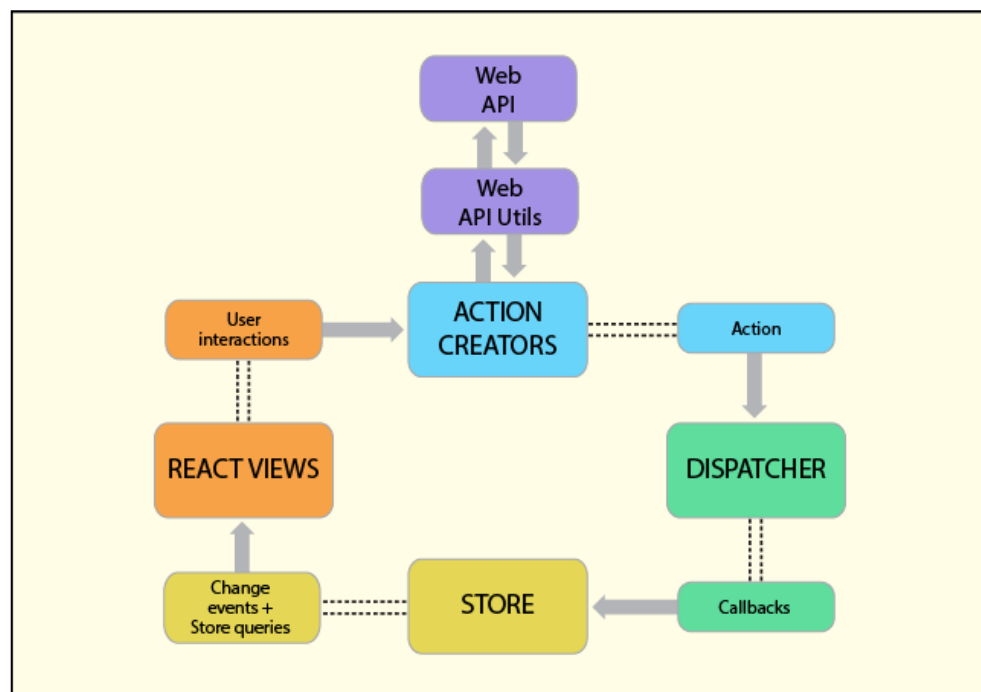


Рисунок 2.3 – Архитектура приложения с React архитектурой

В стандартной архитектуре Flux следующие компоненты:

1. Actions — помощники, которые передают данные в Dispatcher.

2. Dispatcher — получает эти действия и передает полезную нагрузку зарегистрированным callback-ом.

3. Stores — действуют как контейнеры для состояния приложения и логики. Реальная работа приложения происходит в Stores. Stores, зарегистрированные для прослушивания действий Dispatcher, будут соответственно и обновлять View.

4. Controller Views — React компоненты захватывают состояние из Stores, а затем передают дочерним компонентам.

Контроллеры в MVC и Flux различаются. Здесь контроллеры являются Controller-View и находятся на самой вершине иерархии. View — это React компоненты.

Так же в моем курсовом проекте я использовал Apollo Client. Это пакет, позволяющий облегчить работу по получению, обработке и отправке GraphQL запросов на GraphQL сервер. Основной особенностью Apollo Client является несколько типов кэширования информации, что позволяет уменьшить количество запросов на сервер.

Весь функционал, как правило, находится в Store. В Store выполняется вся работа и сообщается Dispatcher, какие события/действия он прослушивает.

Когда происходит событие, Dispatcher отправляет “полезную нагрузку” в Store, который зарегистрирован для прослушивания конкретно этого события. Теперь в Store необходимо обновить View, что в свою очередь вызывает действие. Действие, точно также как и имя, тип события и много другое известны заранее.

View распространяет Action через центральный Dispatcher, и это будет отправлено в различные Stores. Эти Stores будут отображать бизнес-логику приложения и другие данные. Store обновляет все View.

Наиболее хорошо это работает совместно со стилем программирования React, тогда Store отправляет обновления без необходимости подробно описывать, как изменять представления между состояниями.

Это доказывает, что шаблон проектирования Flux следует за однонаправленным потоком данных. Action, Dispatcher, Store и View — независимые узлы с конкретными входными и выходными данными. Данные проходят через Dispatcher, центральный хаб, который в свою очередь управляет всеми данными.

2.2 Разработка диаграммы вариантов использования

UML — это унифицированный графический язык моделирования для описания, визуализации, проектирования и документирования объектно-ориентированных систем. UML призван поддерживать процесс моделирования ПС на основе объектно-ориентированного подхода, организовывать взаимосвязь концептуальных и программных понятий, отражать проблемы масштабирования сложных систем. Модели на UML используются на всех этапах жизненного цикла ПС, начиная с бизнес-анализа и заканчивая сопровождением системы.

Диаграмма вариантов использования веб-ресурса – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Диаграммы вариантов использования применяются при бизнес-анализе для моделирования видов работ, выполняемых организацией, и для моделирования функциональных требований к ПС при ее проектировании и разработке. Построение модели требований при необходимости дополняется их текстовым описанием.

Диаграмму вариантов использования есть смысл строить во время изучения технического задания, она состоит из графической диаграммы, описывающей действующие лица и прецеденты, а также спецификации, представляющего собой текстовое описание конкретных последовательностей действий (потока событий), которые выполняет пользователь при работе с системой.

Спецификация затем станет основой для тестирования и документации, а на следующих этапах проектирования она дополняется и оформляется в виде диаграммы (в рамках ICONIX используется диаграмма последовательности, но в UML для этого имеются также диаграммы деятельности).

На диаграмме использования изображаются:

1. Актеры – группы лиц или систем, взаимодействующих с нашей системой.
2. Варианты использования (прецеденты) – сервисы, которые наша система предоставляет актерам.
3. Комментарии.
4. Отношения между элементами диаграммы.

Сплошные линии на диаграмме представляют собой отношения ассоциации, отражающие возможность использования актером прецедента.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization).

После того, как определен набор вариантов использования, можно приступать к составлению сценариев. Сценарии должны описываться с точки зрения пользователя, при этом важно описывать взаимодействие пользователя с элементами интерфейса.

В результате анализа требований, сформулированных на этапе планирования, была разработана диаграмма вариантов использования. С ее использованием будет проходить дальнейшая разработка функциональных возможностей веб-приложения.

Диаграмма вариантов использования представлена на рисунке 2.4.

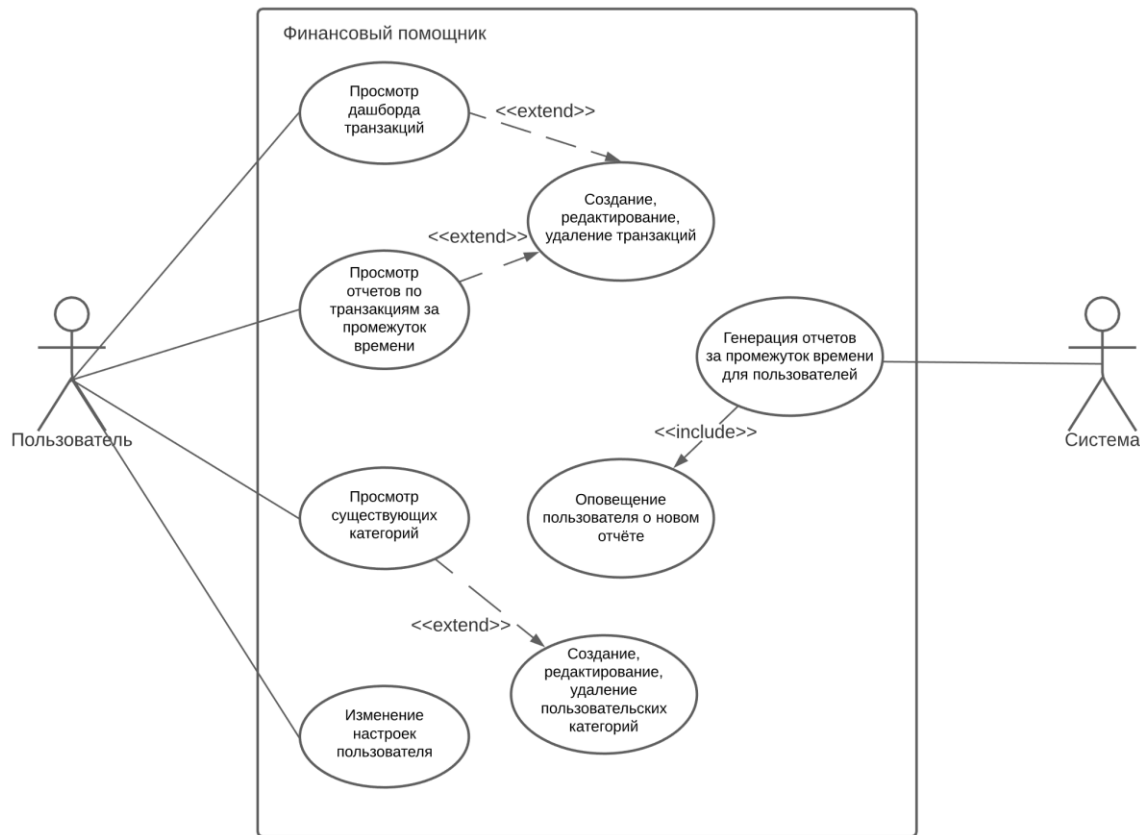


Рисунок 2.4 – Диаграмма вариантов использования

Из диаграммы вариантов использования видно действующих лиц, их взаимодействие с системой и ожидаемую функциональность системы. С ее использованием будет проходить дальнейшая разработка веб-приложения.

2.3 Структура моделей и классов

Как известно, база данных – это хранилище структурированной информации, пассивное по своей сути. Бизнес-логика приложения реализуется где-то вне базы, в виде «набора действий для достижения требуемого результата». В случае внесения изменений в хранимый набор данных результатом должно стать новое состояние базы.

Если предметная область автоматизации представляет собой систему взаимодействующих значений, то ее можно описать ER-моделью.

Диаграмма классов является ключевым элементом редактора UML-диаграмм, поскольку зачастую приложения генерируются именно с диаграммы классов. Диаграмма классов – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании –

описании существующих и используемых систем. Информация с диаграммы классов напрямую отображается в исходный код приложения. Таким образом, диаграмма классов – конечный результат проектирования и отправная точка процесса разработки.

В соответствии с моделью базы данных и поставленными задачами был разработан пример модели классов структуры доступа к базе данных, представленная на рисунке 2.5.

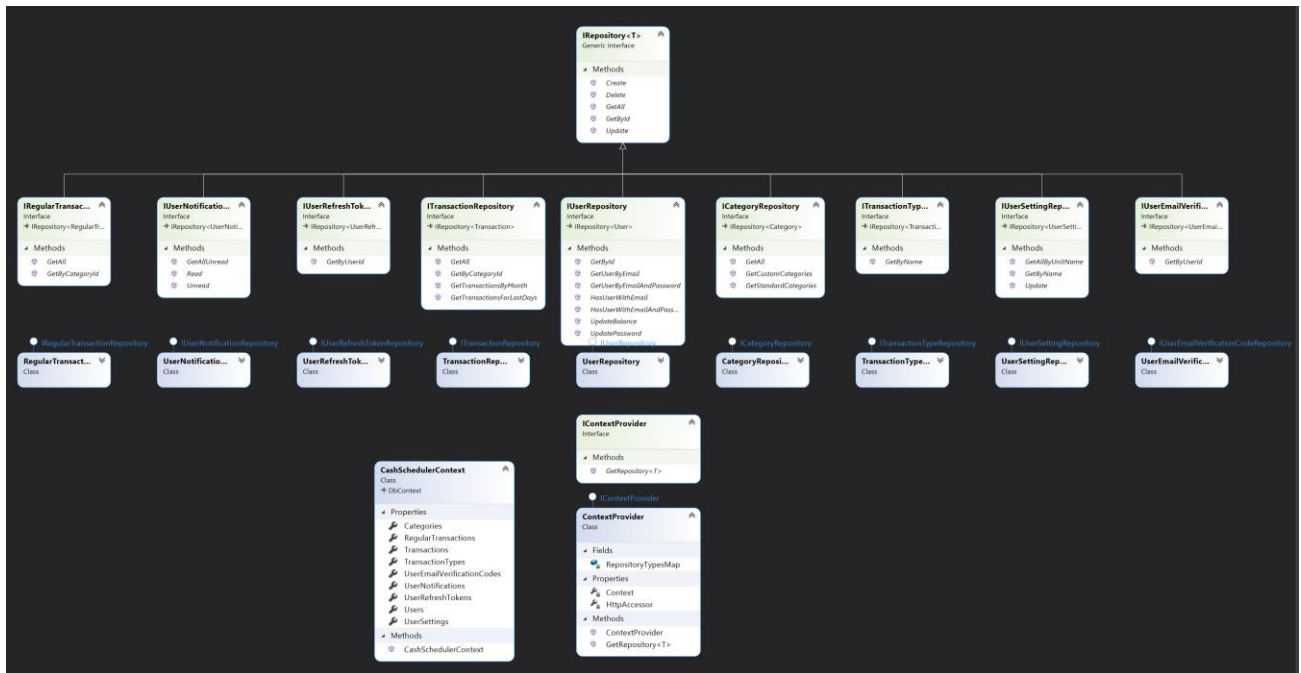


Рисунок 2.5 – Модель классов репозиторий базы данных

На диаграмме выше показана структура репозиторий и реализация паттерна проектирования “Unit Of Work”, при котором у нас доступ к репозиториям предоставляется через специально созданный для этого класс.

Такая модель не только образует логическую структуру базы данных, но обладает всеми свойствами программы, каковой, по сути, и является – в вычислительной среде, образованной методами упомянутых абстрактных сущностей.

2.4 Структура курсового проекта

Разработанный веб-ресурс построен по архитектуре клиент-сервер, которые взаимодействуют по протоколу HTTP.

Такая архитектура была выбрана с целью максимального отделения представления от бизнес-логики, то есть в том, что любое пользовательское приложение вначале делится на два модуля – один из которых отвечает за

реализацию собственно самой бизнес логики (Модель), а второй – за взаимодействие с пользователем (Пользовательский Интерфейс или Представление).

Серверная часть была написана с помощью фреймворка ASP .NET Core MVC. Платформа ASP.NET Core MVC представляет собой фреймворк для создания сайтов и веб-приложений с помощью реализации паттерна MVC.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью.

При разработке серверной части веб-ресурса были созданы проекты на основе MVC архитектуры. Структура проекта представлена на рисунке 2.6.

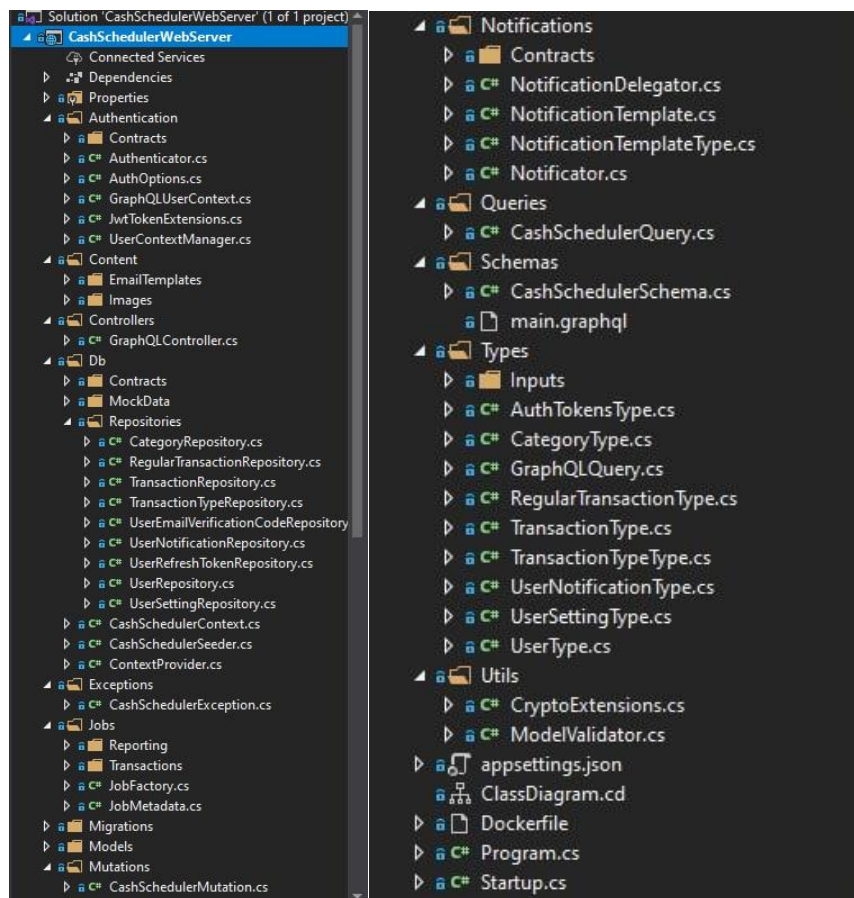


Рисунок 2.6 – Структура серверной части веб-ресурса

Для написания клиентской части веб-ресурса использовался фреймворк React [3] и Apollo Client [2]. Файл со всеми используемыми модулями находится в приложении Б.

React – это инструмент для создания пользовательских интерфейсов. Его главная задача – обеспечение вывода на экран того, что можно видеть на веб-страницах. React значительно облегчает создание интерфейсов благодаря

разбиению каждой страницы на небольшие фрагменты. Мы называем эти фрагменты компонентами.

Разработанная с помощью вышеописанных технологий клиентская часть имеет структуру, представленную на рисунке 2.7.

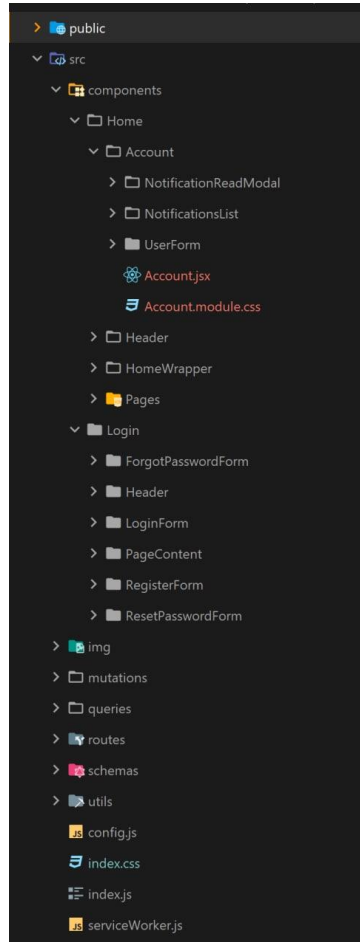


Рисунок 2.7 – Структура клиентской части веб-ресурса

Компонентно-ориентированный подход, возможность с легкостью изменять имеющиеся компоненты и переиспользовать код превращают React разработку в непрерывный процесс улучшения. Таким образом, ничто не мешает использовать их снова и снова в проектах разного типа. Возможность с легкостью заново использовать уже имеющийся код повышает скорость разработки, упрощает процесс тестирования, и, как результат, понижает затраты.

3 Разработка программного средства

На основе сформированных требований разрабатывается база данных для того, чтобы определить состав информации, которая должна находиться в базе данных для решения всего комплекса задач предметной области, а также выявить логические взаимосвязи данных, размещаемых в БД. Состав и взаимосвязи данных должны быть отображены моделью данных предметной области. На основе этой модели необходимо определить соответствующую ей логическую структуру базы данных для выбранной СУБД. Процесс разработки необходимо осуществлять в соответствии с концепцией логической организации данных выбранной СУБД.

3.1 Описание функциональных возможностей приложения

Разрабатываемый веб-ресурс представляет собой веб-ресурс, позволяющий удобно контролировать жизненный цикл IT проекта. В результате была создана база данных, которая удовлетворяет поставленным задачам.

Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных.

Функционально программное средство должно выполнять следующие задачи:

- сохранение рабочей информации в централизованной базе данных;
- регистрация одиночных и повторяющихся денежных транзакций;
- возможность получения оповещений об отчетах бюджета за месяц;
- анализ бюджета и транзакций на основе расходов и доходов за промежуток времени;
- настройка параметров работы приложения для пользователя.

3.2 Разработка базы данных

Для разработки и управления базой данных курсового проекта использовалась объектно-реляционная система управления базами данных «MS SQL Server».

Выбор данного системного продукта произошел по нескольким очевидным причинам: легкая инсталляция, бесплатна для разработки, раннее полученные знания по использованию данной базы данных.

База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной

вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных.

Microsoft SQL Server [5] – система управления реляционными базами данных, разработанная корпорацией Microsoft. Основным используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Для базы данных было разработано 9 таблиц.

Логическая схема базы данных со всеми таблицами и связями представлена в приложении А.

На рисунке 3.1 представлена структура таблицы Categories, которая содержит в себе категории транзакций.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Name	nvarchar(30)	<input type="checkbox"/>
TypeName	nvarchar(450)	<input checked="" type="checkbox"/>
CreatedById	int	<input checked="" type="checkbox"/>
IsCustom	bit	<input type="checkbox"/>
IconUrl	nvarchar(MAX)	<input checked="" type="checkbox"/>

Рисунок 3.1 – Таблица Categories

Таблица включает поля:

- Id – первичный ключ, идентификатор категории;
- Name – название категории;
- Type Name – название типа категории;
- CreatedById – вторичный ключ, идентификатор пользователя;
- IsCustom – булевская переменная, отвечающая за определение стандартной или пользовательской категорией является;
- IconUrl – url иконки категории.

На рисунке 3.2 представлена структура таблицы RegularTransactions, которая содержит в себе транзакции, которые повторяются через определенный интервал времени.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Title	nvarchar(30)	<input checked="" type="checkbox"/>
CreatedById	int	<input checked="" type="checkbox"/>
TransactionCategoryId	int	<input checked="" type="checkbox"/>
Amount	float	<input type="checkbox"/>
Date	datetime2(7)	<input type="checkbox"/>
NextTransactionDate	datetime2(7)	<input type="checkbox"/>
Interval	nvarchar(MAX)	<input type="checkbox"/>

Рисунок 3.2 – Таблица RegularTransactions

Таблица включает поля:

- Id – первичный ключ, идентификатор регулярной транзакции;
- Title – заголовок регулярной транзакции;
- CreatedById – вторичный ключ, идентификатор пользователя;
- TransactionCategoryId – вторичный ключ, идентификатор категории;
- Amount – количество денег за регулярную транзакцию;
- Date – дата создания регулярной транзакции;
- NextTransactionDate – дата следующей регулярной транзакции;
- Interval – интервал регулярной транзакции.

На рисунке 3.3 представлена структура таблицы Transactions, которая содержит транзакции .

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Title	nvarchar(30)	<input checked="" type="checkbox"/>
CreatedById	int	<input checked="" type="checkbox"/>
TransactionCategoryId	int	<input checked="" type="checkbox"/>
Amount	float	<input type="checkbox"/>
Date	datetime2(7)	<input type="checkbox"/>

Рисунок 3.3 – Таблица Transactions

Таблица включает поля:

- Id – первичный ключ, идентификатор транзакции;
- Title – заголовок транзакции;
- CreatedById – вторичный ключ, идентификатор пользователя;
- TransactionCategoryId – вторичный ключ, идентификатор категории;
- Amount – количество денег за транзакцию;
- Date – дата создания транзакции.

На рисунке 3.4 представлена структура таблицы TransactionTypes, которая содержит информацию о возможных типах транзакций.

Column Name	Data Type	Allow Nulls
Name	nvarchar(450)	<input type="checkbox"/>
IconUrl	nvarchar(MAX)	<input checked="" type="checkbox"/>

Рисунок 3.4 – Таблица TransactionTypes

Таблица включает поля:

- Name – первичный ключ, название типа транзакции;
- IconUrl – ссылка на иконку для типа транзакции.

На рисунке 3.5 представлена структура таблицы UserEmailVerificationCodes, которая содержит информацию о кодах для подтверждения смены пароля на почту.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Code	nvarchar(MAX)	<input type="checkbox"/>
ExpiredDate	datetime2(7)	<input type="checkbox"/>
UserId	int	<input type="checkbox"/>

Рисунок 3.5 – Таблица UserEmailVerificationCodes

Таблица включает поля:

- Id – первичный ключ, идентификатор кода для смены пароля;
- Code – код;
- ExpiredDate – время истечения токена;
- UserId – вторичный ключ, идентификатор пользователя.

На рисунке 3.6 представлена структура таблицы UserNotifications, которая содержит оповещения.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Title	nvarchar(50)	<input type="checkbox"/>
[Content]	nvarchar(MAX)	<input type="checkbox"/>
CreatedForId	int	<input checked="" type="checkbox"/>
IsRead	bit	<input type="checkbox"/>

Рисунок 3.6 – Таблица UserNotifications

Таблица включает поля:

- Id – первичный ключ, идентификатор оповещений;
- Title – заголовок оповещения;
- Content – тело оповещения;
- IsRead – показывает было ли сообщение прочитано пользователем;
- CreatedForId – вторичный ключ, идентификатор пользователя.

На рисунке 3.7 представлена структура таблицы UserRefreshTokens, которая содержит токен для обновления пароля.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Token	nvarchar(MAX)	<input type="checkbox"/>
ExpiredDate	datetime2(7)	<input type="checkbox"/>
UserId	int	<input type="checkbox"/>

Рисунок 3.7 – Таблица UserRefreshTokens

Таблица включает поля:

- Id – первичный ключ, идентификатор токена;
- Token – токен;
- ExpiredDate – время истечения токена;
- UserId – вторичный ключ, идентификатор пользователя.

На рисунке 3.8 представлена структура таблицы Users, которая содержит в себе информацию о пользователях.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
FirstName	nvarchar(50)	<input checked="" type="checkbox"/>
LastName	nvarchar(50)	<input checked="" type="checkbox"/>
Email	nvarchar(MAX)	<input type="checkbox"/>
Password	nvarchar(MAX)	<input type="checkbox"/>
Balance	float	<input type="checkbox"/>

Рисунок 3.8 – Таблица Users

Таблица включает поля:

- Id – первичный ключ, идентификатор пользователя;
- FirstName – имя пользователя;
- LastName – фамилия пользователя;
- Email – почта, уникальное имя пользователя;
- Password – пароль пользователя;
- Balance – баланс пользователя.

На рисунке 3.9 представлена структура таблицы UserSettings, которая содержит информацию о настройках пользователей.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Name	nvarchar(MAX)	<input type="checkbox"/>
Value	nvarchar(MAX)	<input checked="" type="checkbox"/>
UnitName	nvarchar(MAX)	<input type="checkbox"/>
SettingForId	int	<input checked="" type="checkbox"/>

Рисунок 3.9 – Таблица UserSettings

Таблица включает поля:

- Id – первичный ключ, идентификатор настройки;
- Name – название настройки;
- Value – текущее значение настройки;
- UnitName – название категории настройки;
- SettingForId – идентификатор пользователя с этой настройкой.

3.3 Описание алгоритма смены пароля пользователя

Если пользователь забыл пароль с помощью которого можно войти в личный кабинет, он может восстановить его на начальной странице пройдя через несколько шагов.

При попытке восстановить доступ к аккаунту, первым делом, пользователь должен ввести свою почту, которую он указал при регистрации. В будущем на эту почту будет отправлен специальный код подтверждения, который пользователю надо предоставить, чтобы получить возможность изменить пароль.

При вводе пользователем почты, отправляется запрос на сервер, где происходит поиск пользователя с такой почтой в базе данных. Если такой пользователь был найден, то ему отправляется сообщение со случайно сгенерированным кодом на почту и, в добавок, это же оповещение дублируется на аккаунт пользователя. Чтобы пользователь, находясь уже залогиненным в системе с другого устройства, мог посмотреть код доступа для смены пароля. Если пользователя с предоставленной почтой не было найдено, в ответе возвращается ошибка.

После того, как пользователь получил код и ввел его в поле на форме, отправляется еще один запрос. На сервере код, полученный от пользователя и код, записанный в базе данных для этого пользователя сравниваются, а так же проверяется время жизни этого кода, которое составляет 5 минут. Если пользователь не успел ввести код за это время, ему придется запросить еще один такой код.

Если код валидный, на интерфейсе пользователю предоставляется возможность указать и подтвердить новый пароль для аккаунта. В таком случае проверка на валидность введенного в предыдущем шаге токена совершается еще

раз, чтобы убедиться, чтобы не было возможности сменить пароль через веб-апи, без прохождения верификации кодом по почте.

В конечном итоге, если все в порядке, то у пользователя хэшируется введенный им новый пароль и его перекидывает на форму логина, где он может применить только что измененные данные входа.

Листинг серверной логики для описанного выше процесса приведен в приложении В.

3.4 Процесс создания транзакций

Работа с API веб сервера происходит с помощью GraphQL. Создание транзакций проходит в несколько этапов.

Первым делом проверяется доступ у текущего пользователя по токenu доступа, который берется из Authorization хедера запроса. Если токен валидный и имеет доступ для выполнения метода создания транзакции, то после этого валидируется экземпляр транзакции, полученный из запроса.

На поля модели транзакции наложены различного рода ограничения и если эти какое-либо из полей не проходит валидацию, то происходит исключение, с информацией о том, какое поле было некорректно заполнено.

Транзакции назначается пользователь исходя из токена, полученного из запроса. Каждый токен хранит в себе идентификатор пользователя, что позволяет нам определять, какой токен соответствует какому пользователю.

Если все пункты выше в порядке, происходит запись транзакции в базу данных. После чего вызывается метод обновления баланса у пользователя, который изменяет его текущий баланс, если транзакция была сделана в прошлом времени или сегодня. Если транзакция была назначена с датой в будущем времени, баланс пользователя не изменится.

После завершения всех пунктов выше в ответе возвращается экземпляр только что созданной транзакции.

Создание повторяющихся транзакций происходит похожим образом за тем исключением, что они никак не влияют на баланс в текущий момент времени, т.к. могут быть созданы с датой только в будущем времени. Такие транзакции обрабатываются отдельно, чтобы обеспечить синхронизацию баланса согласно датам, проставленным в этих транзакциях.

Пример метода отвечающего за создание транзакций в приложении приведен в листинге 3.1.

```

        public async Task<Transaction> Create(Transaction transaction)
        {
            ModelValidator.ValidateModelAttributes(transaction);
            transaction.CreatedBy =
            ContextProvider.GetRepository<IUserRepository>().GetById((int)UserId);
            transaction.TransactionCategory =
            ContextProvider.GetRepository<ICategoryRepository>().GetById(transaction.CategoryId);
            if (transaction.TransactionCategory == null)
            {
                throw new CashSchedulerException("There is no such category", new string[] { "categoryId" });
            }
            Context.Transactions.Add(transaction);
            await Context.SaveChangesAsync();
            await
            ContextProvider.GetRepository<IUserRepository>().UpdateBalance(transaction, null, isCreate: true);

            return GetById(transaction.Id);
        }

```

Листинг 3.1 – Метод Create() репозитория транзакций

В данном методе функция `ModelValidator.ValidateModelAttributes()` является методом проверки валидности модели. Как и объяснялось выше, здесь происходит получение текущего пользователя по его токenu авторизации, присваивание транзакции ссылки на этого пользователя в базе данных, а так же ссылки на категорию транзакции, айди которой был предоставлен в запросе. После чего происходит сохранение транзакции и обновление баланса пользователя. Последним шагом стоит возврат только что созданной транзакции в ответе.

4 Руководство программиста

В данной главе будут представлены шаги, описывающие как развернуть приложение. Разворачивать приложение будем при помощи Docker.

Docker [6] — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуском кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Нам понадобится Docker последней версии. В Docker Hub заранее были загружены images, поэтому понадобится лишь загрузить их при помощи команд представленных в листинге 4.1:

```
docker pull ilyamatsuev/cash-scheduler-server
docker pull ilyamatsuev/cash-scheduler-client
```

Листинг 4.1 – Команды Docker

Далее нужно составить файл `docker-compose.yml` (представлен в приложении Д) с заранее определенными сервисами и параметрами для них. Сервисы представляют выгруженные клиент и сервер, а третий представляет базу данных. Запуск производится с помощью команды представленной в листинге 4.2.

```
docker-compose up
```

Рисунок 4.2 – Команда запуска всех трех сервисов из файла compose

После этого будут запускаться каждый из сервисов, описанных в файле `docker-compose.yml`. При необходимости будут докачиваться докер имейджи, используемые как зависимости в имейджах данного проекта. После того как все контейнеры инициализируются, приложение будет доступно для клиентов по адресу `http://localhost:3000/`.

5 Руководство пользователя

Данная глава содержит описание некоторых функций приложения для более легкого восприятия конечного пользователя.

5.1 Главная страница

На главной странице продемонстрирован персональный календарь. В окошках на каждый день отображается сумма доходов за этот день (зеленый цвет), расходов за день (красный), запланированный доход (синий), запланированный расход (желтый). Все это продемонстрировано на рисунке 5.1.

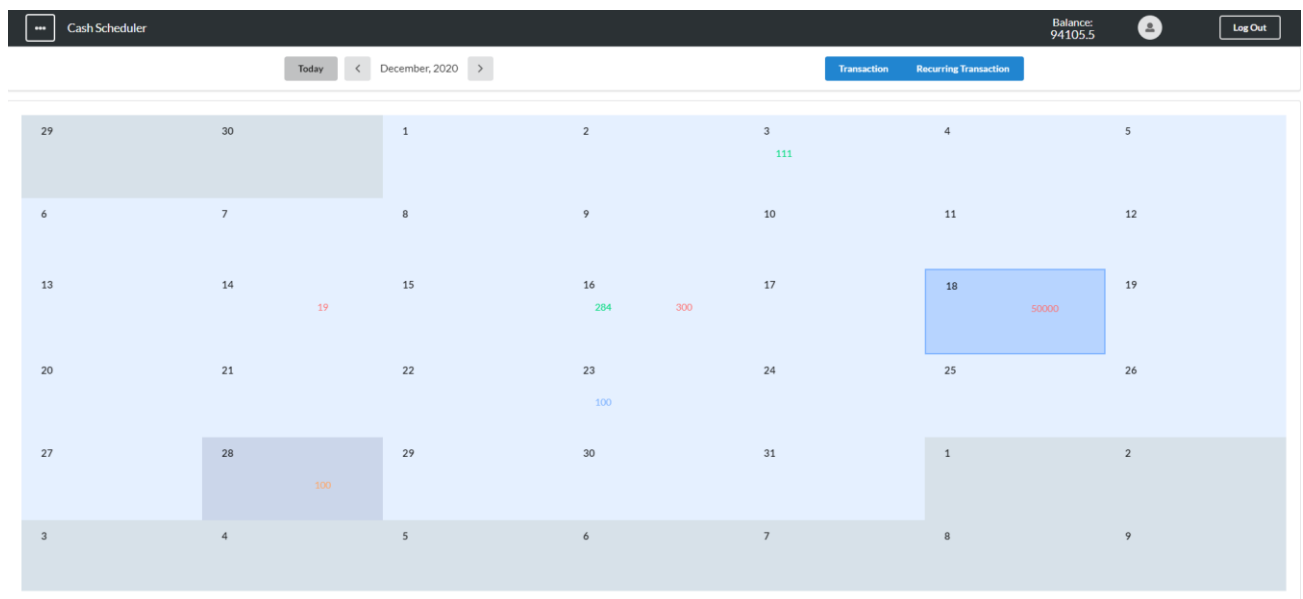


Рисунок 5.1 – Главная страница приложения

Сверху есть кнопки для переключения по месяцам и возвращения к текущему месяцу. Так же в верхнем левом углу присутствует кнопка для открытия меню страниц.

5.2 Добавление единичной транзакции

Для того чтобы добавить единичную транзакцию пользователю необходимо на главной странице нажать на кнопку «Transaction». После нажатия на эту кнопку пользователь увидит форму, которая показана на рисунке 5.2.

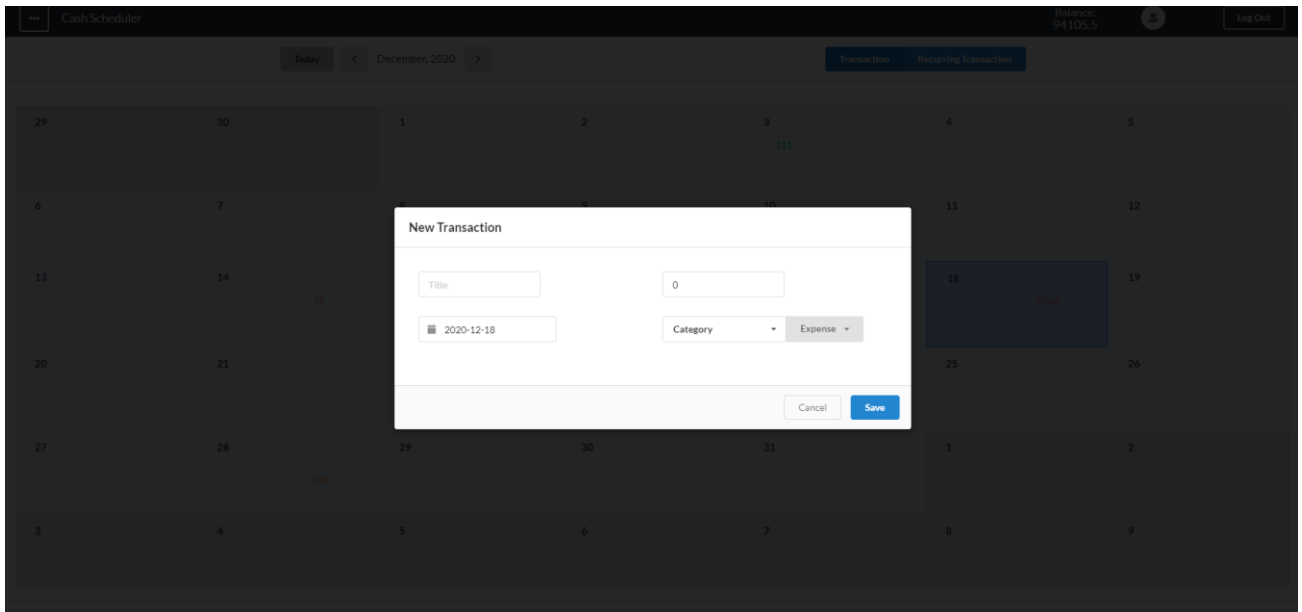


Рисунок 5.2 – Форма для добавления транзакции

Здесь можно выбрать вид транзакции расход это, либо доход, в зависимости от этого будет выбор категории. Все возможные категории продемонстрированы на рисунке 5.3.

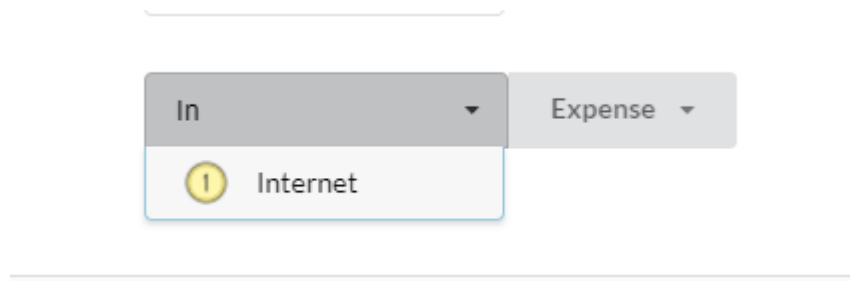


Рисунок 5.3 – Категории

Так же в форме существует возможность добавления повторяющейся транзакции (рисунок 5.4), которая будет автоматически повторяться через указанный пользователем интервал (рисунок 5.5).

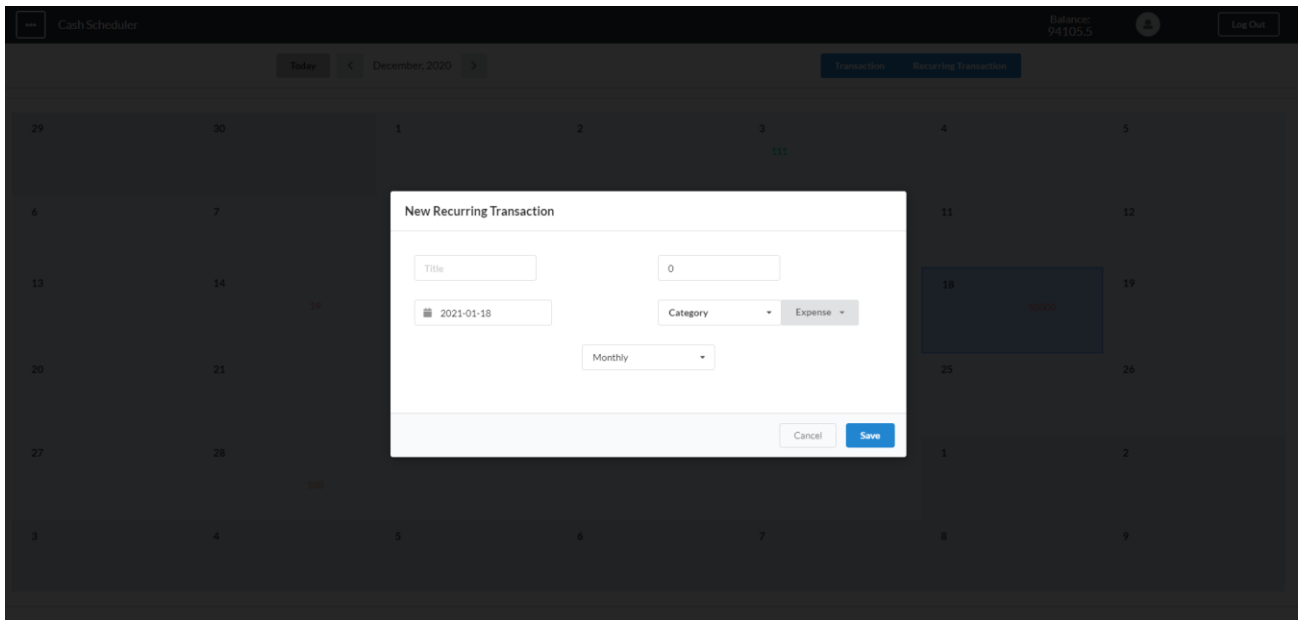


Рисунок 5.4 – Добавление повторяющейся транзакции

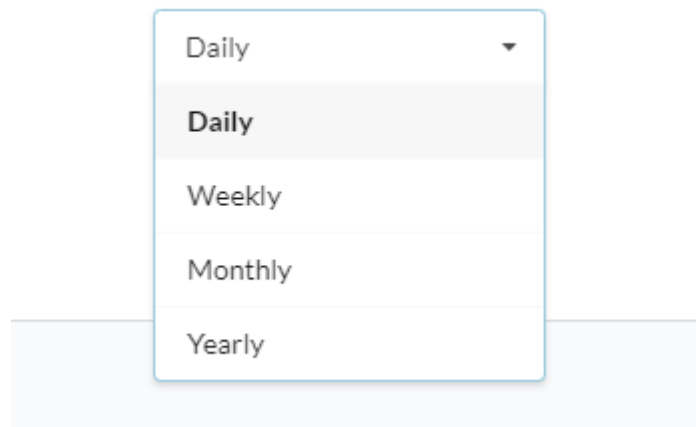


Рисунок 5.5 – Интервал

В зависимости от того, какой выбран интервал, автоматически будет создаваться транзакция каждый день/неделю/месяц/год. И пользователю не нужно делать это вручную.

5.3 Секция оповещений

В данном разделе продемонстрированы оповещения. Чтобы увидеть эту секцию, необходимо проставить соответствующую настройку на странице настроек, а затем кликнуть на значёк аккаунта. Здесь справа отображаются оповещения связанные с отчётами на основе расходов и доходов пользователя и вообще любая полезная информация, которую пользователь имеет право

получать. Пользователь сам отправлять ничего не может, он только получает оповещения (рисунок 5.6).

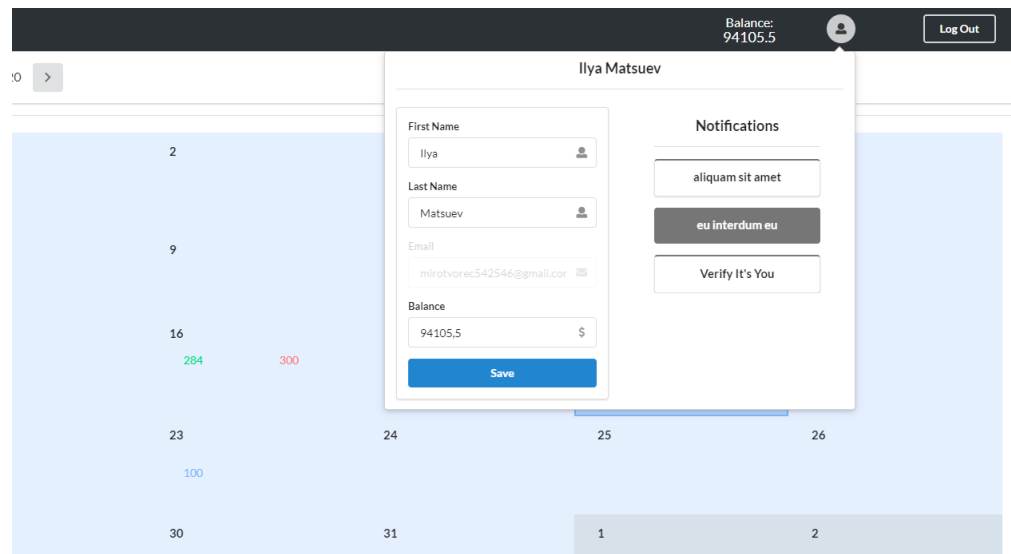


Рисунок 5.6 – Секция оповещений

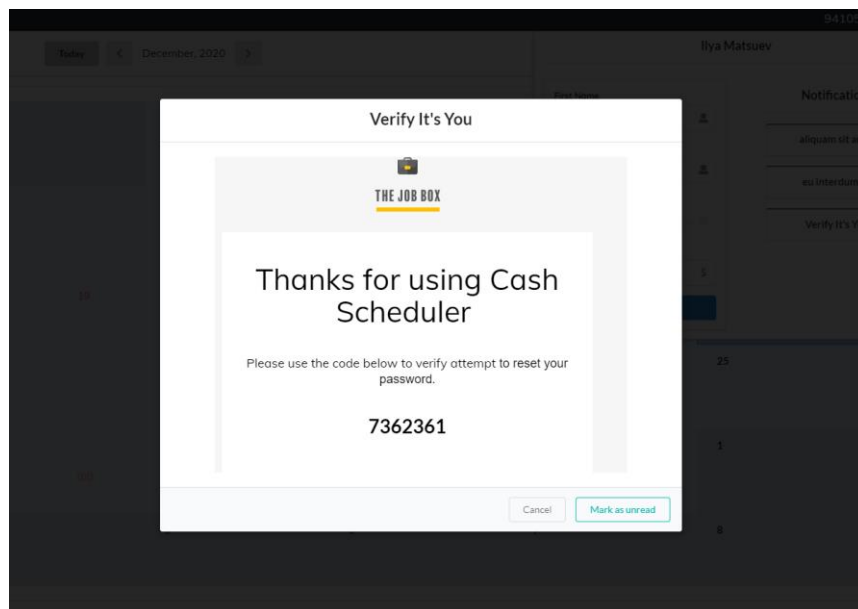


Рисунок 5.7 – Открытое оповещение

Если сообщение является непрочитанным, оно будет помечено сервым цветом. На рисунке 5.7 показано как выглядит открытое сообщение. Пользователь так же, сам может пометить оповещение, как непрочитанное, чтобы изучить чуть позже.

5.4 Страница транзакций

Чтобы перейти на данную страницу просто нужно нажать в меню слева на раздел «Transactions». На этой странице пользователь может просматривать отчеты трат, доходов и их соотношения по месяцам (рисунок 5.8).

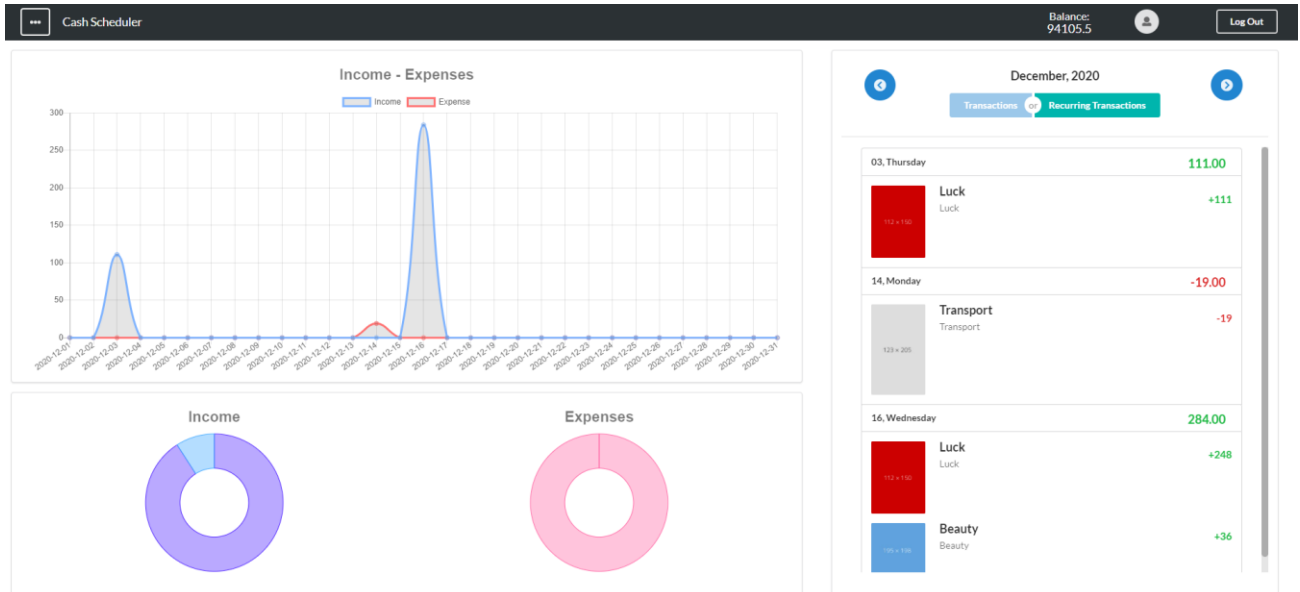


Рисунок 5.8 – Страница транзакций

Можно посмотреть, как много было потрачено или заработано на определенную категорию за выбранный месяц. Это можно увидеть на графике внизу страницы, показано на рисунке 5.9.

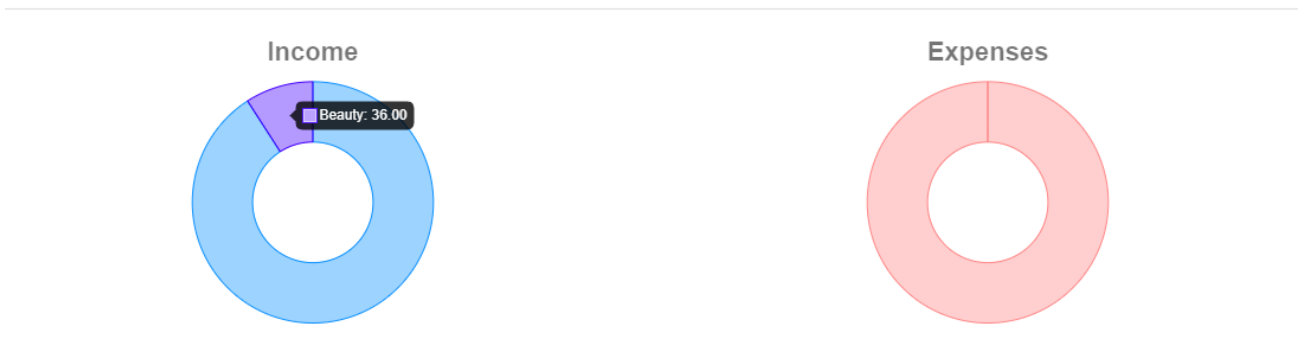


Рисунок 5.9 – Соотношение расходов и доходов по категориям

Следующий же график показывает траты и доходы за каждый день выбранного месяца. Он находится вверху страницы транзакций, пример его продемонстрирован на рисунке 5.10.

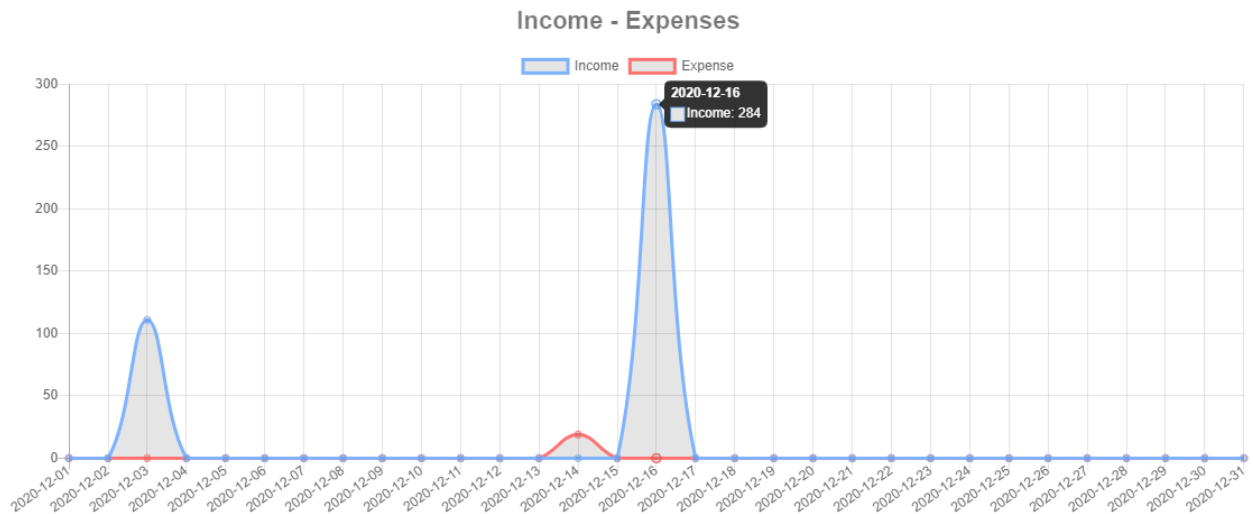


Рисунок 5.10 – Количество расходов и доходов по дням

На странице транзакций справа можно увидеть список транзакций за выбранный месяц сгруппированный по дням месяца.

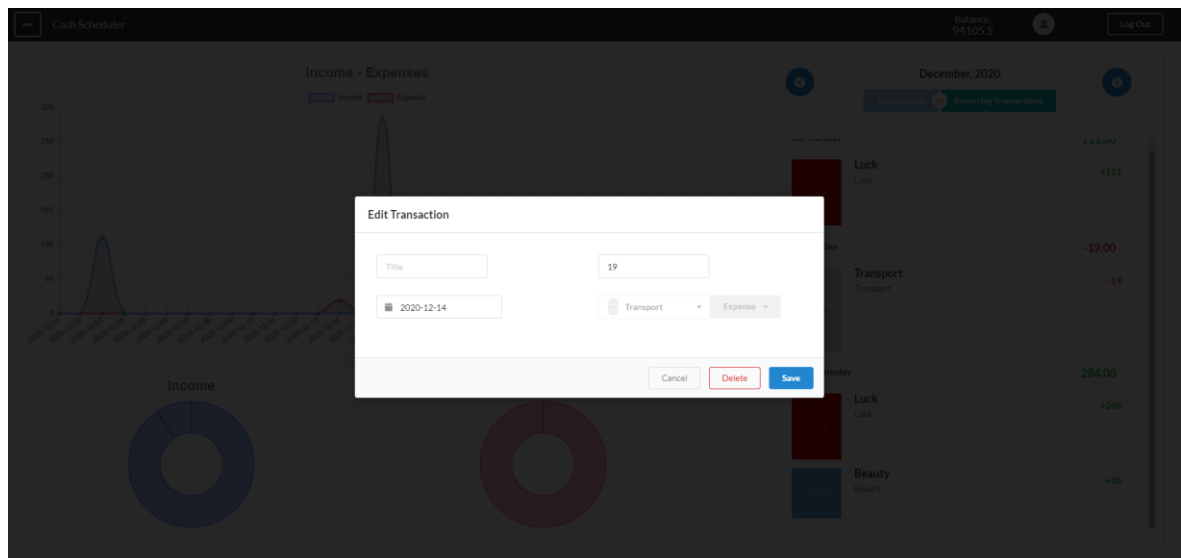


Рисунок 5.11 – Редактирование транзакций

Реализована возможность для пользователя выбрать какую-либо транзакцию из списка и отредактировать либо ее заголовок, либо сумму, либо дату, изменить категорию, к которой относится транзакция, нельзя, однако предусмотрено удаление транзакции (рисунок 5.11).

5.5 Страница категорий

Чтобы перейти на данную страницу просто нужно нажать в меню слева на раздел «Categories». Здесь можно увидеть, как стандартные, так и пользовательские категории. К стандартным относятся: salary, Gift, Pets, Transport, Spot и др. Более сложными по своему функционалу являются пользовательские категории, которые показаны на рисунке 5.12.

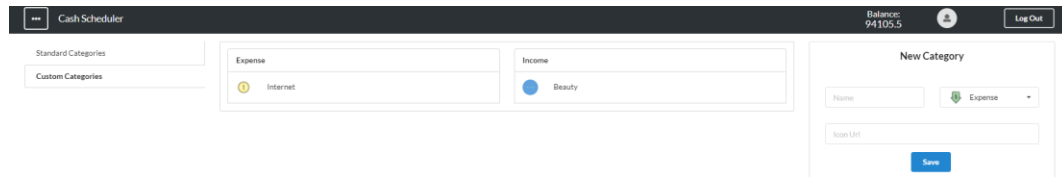


Рисунок 5.12 – Пользовательские категории

Пользовательские категории можно редактировать, добавлять свои и удалять, тем самым стандартные категории изменить невозможно. Пример добавления новой пользовательской категории продемонстрирован на рисунке 5.13.

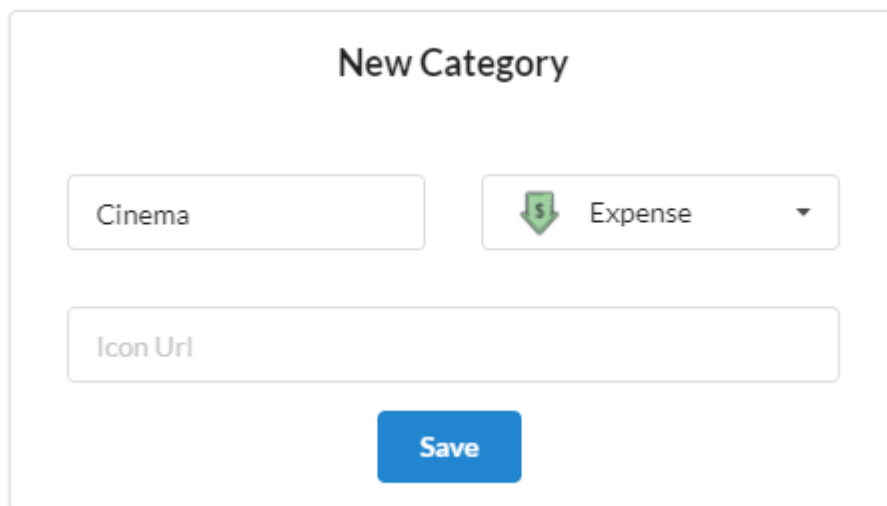


Рисунок 5.13 – Добавление пользовательской категории

Так же доступна возможность редактирования и удаления категорий, что продемонстрировано на рисунке 5.14.

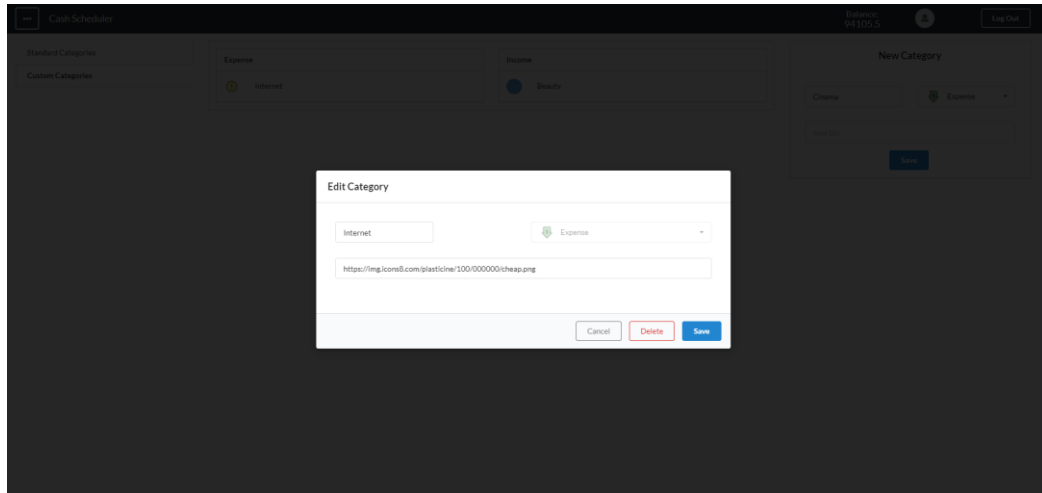


Рисунок 5.14 – Изменение пользовательской категории

При этом при попытке удаления категории, пользователь будет осведомлен, что в таком случае будут удалены все транзакции привязанные к данной категории.

5.6 Страница настроек

Чтобы перейти на данную страницу просто нужно нажать в меню слева на раздел «Settings». На этой странице реализовано три раздела: general, notifications, developers. В первом разделе предусмотрено включение и выключение отображения баланса, если включить эту функцию, то баланс пользователя будет отображаться сверху слева (рисунок 5.15).

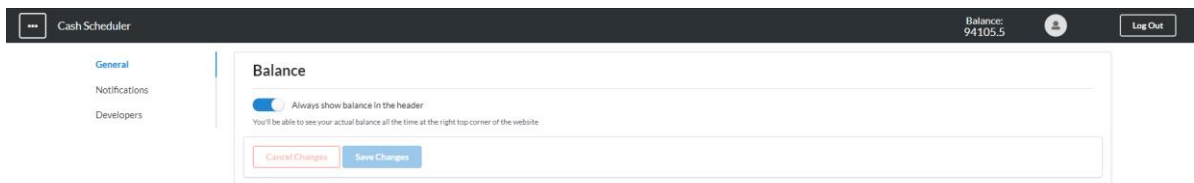


Рисунок 5.15 – Добавление пользовательской

Все настройки хранятся в базе данных, поэтому если обновить страницу, изменения сохраняются.

Во втором разделе настроек есть такие настройки как включение уведомлений, возможность получать копию уведомлений на почту, включение и выключение звука в браузере при оповещении (рисунок 5.16).

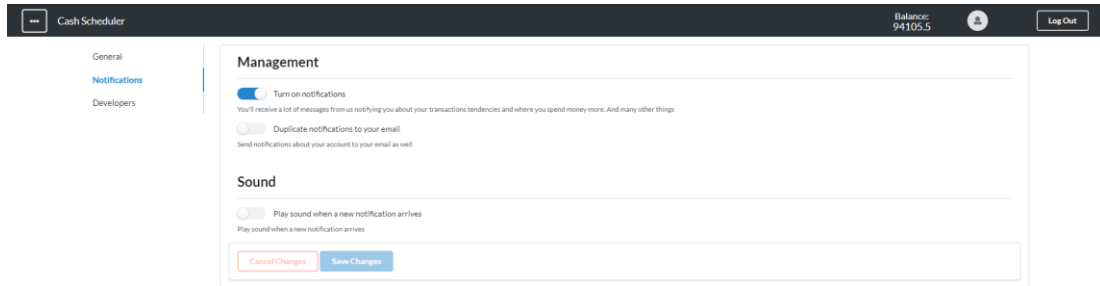


Рисунок 5.16 – Раздел настроек «Notifications»

Третий раздел настроек разработан специально для разработчиков. В нем имеется кнопка, которая копирует в буфер обмена токен доступа к вашему аккаунту, чтобы вы могли работать со своим API через другие приложения. Пример этого показан на рисунке 5.17.

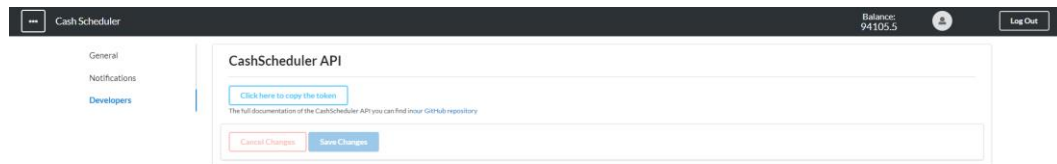


Рисунок 5.17 – Раздел настроек «For developers»

Если пользователь случайно изменил какие-то настройки, то он может с легкостью вернуть предыдущие, нажав на кнопку “Cancel Changes”.

6 Тестирование

До запуска приложения в производство, когда оно станет доступно пользователям, важно убедиться, что данное приложение функционирует, как и должно, что в нем нет ошибок, которые могли бы полностью заблокировать основные сценарии взаимодействия с приложением.

Для обеспечения корректности работы программы, обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

В курсовом проекте организована валидация полей на случай неочевидного или неправильного поведения пользователя.

Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

При регистрации обрабатываются вводимые данные пользователя. Так как поле «email» является уникальным, в базе данных не могут храниться пользователи с одинаковыми логинами, если в базе данных уже существует пользователь с таким же логином, то клиента уведомят об ошибке, это продемонстрировано на рисунке 6.1.

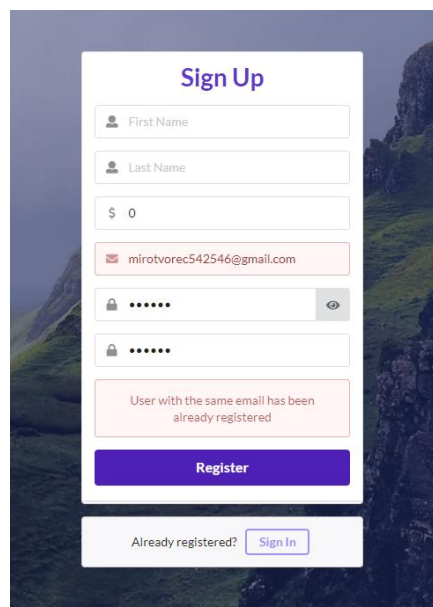
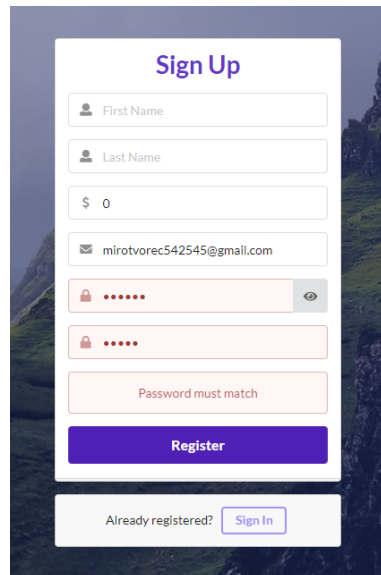
The image shows a 'Sign Up' form with a light purple header. The form contains several input fields: 'First Name', 'Last Name', a currency field with '\$ 0', an email field with 'mirotvorec542546@gmail.com', and two password fields. The second password field has a red error message below it: 'User with the same email has been already registered'. At the bottom of the form is a purple 'Register' button. Below the form is a link that says 'Already registered? Sign In'.

Рисунок 6.1 – Некорректный логин

Также возникают ошибки, если пользователь пытается ввести неверные данные в поле «Confirm password» (рисунок 6.2). Введенный пароль в поле

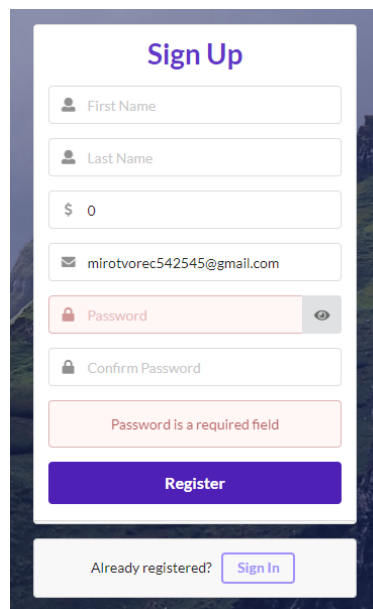
«Password» должен полностью совпадать с введенным паролем в поле «Confirm password».



The screenshot shows a 'Sign Up' form with the following fields: First Name, Last Name, a currency field with '\$ 0', an email field with 'mirotvorec542545@gmail.com', and two password fields. The first password field contains red dots, and the second password field contains four red dots. A red error message 'Password must match' is displayed below the password fields. A purple 'Register' button is at the bottom of the form, and a 'Sign In' link is at the very bottom.

Рисунок 6.2 – Несовпадение паролей

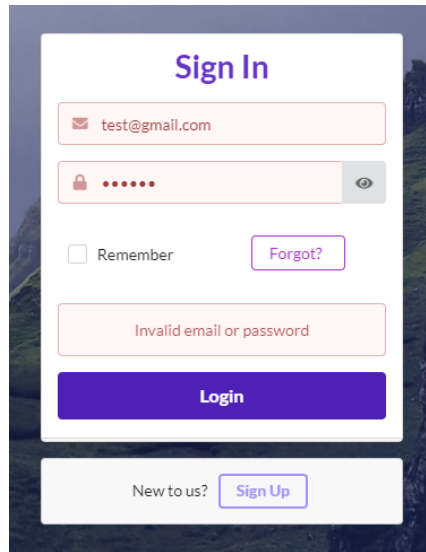
Существует проверка на пустые поля. Если пользователь не ввел данные в поля, система так же уведомит его об этом. Пример не введенных данных в поля «Password» и «Confirm Password» продемонстрирован на рисунке 6.3.



The screenshot shows the same 'Sign Up' form as in Figure 6.2, but with the 'Password' and 'Confirm Password' fields empty. A red error message 'Password is a required field' is displayed below the 'Confirm Password' field. The 'Register' button and 'Sign In' link are still present.

Рисунок 6.3 – Пустые поля

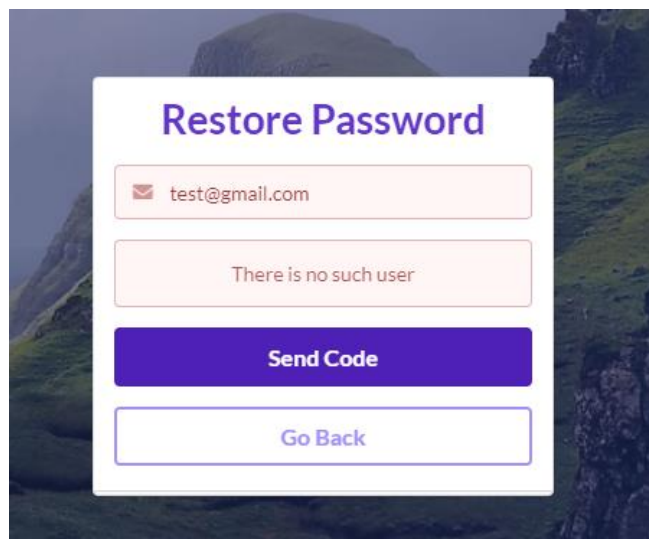
При входе в систему также обрабатываются вводимые данные пользователя. Если пользователь неверно ввел данные в поля для входа, его уведомят об этом, пример уведомления об ошибке предоставлен на рисунке 6.4.



The image shows a 'Sign In' form with a white background and a purple border. At the top, the title 'Sign In' is in purple. Below it, there are two input fields: the first contains 'test@gmail.com' and the second contains six dots. To the right of the password field is an eye icon. Below the fields are two links: 'Remember' with a checkbox and 'Forgot?'. A red error message 'Invalid email or password' is displayed in a red box. At the bottom, there is a purple 'Login' button and a 'New to us? Sign Up' link.

Рисунок 6.4 – Неверно введены данные

В данной курсовой работе разработана функция для тех пользователей которые забыли пароль. В этом разделе так же предусмотрена проверка на вводимые данные. Поле для логина является уникальным поэтому создана проверка на его существование. Соответственно если введенного логина нет в базе данных восстановить данные не удастся, пример работы предоставлен на рисунке 6.5.



The image shows a 'Restore Password' form with a white background and a purple border. At the top, the title 'Restore Password' is in purple. Below it, there is an input field containing 'test@gmail.com'. A red error message 'There is no such user' is displayed in a red box. At the bottom, there are two buttons: a purple 'Send Code' button and a white 'Go Back' button with a purple border.

Рисунок 6.5 – Наличие пользователя в базе данных

Следующим этапом проверки является проверка на правильно введенный код подтверждения для восстановления пароля (рисунок 6.6).

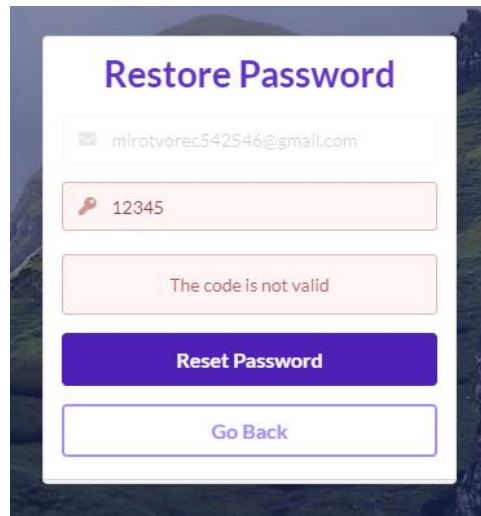


Рисунок 6.6 – Неверный код подтверждения

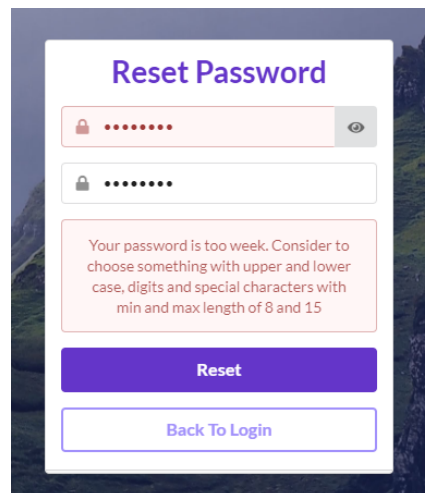


Рисунок 6.7 – Недостаточно сложный пароль

На рисунке 6.7. представлена проверка на достаточной сложности пароль. Пароль должен содержать от 8 до 15 символов, должен включать цифры, буквы нижнего и верхнего регистров и спец символы.

Заключение

В данном курсовом проекте была создана база данных и приложение «Планировщик бюджета», основная функциональность поможет правильно распоряжаться своими деньгами, экономить, следить за доходами и расходами. Были выполнены следующие задачи:

- сохранение рабочей информации в централизованной базе данных;
- регистрация одиночных и повторяющихся денежных транзакций;
- возможность получения оповещений об отчетах бюджета за месяц;
- анализ бюджета и транзакций на основе расходов и доходов за промежуток времени;
- настройка параметров работы приложения для пользователя.

Данное программное средство использует технологии ASP .NET Core для разработки серверной части и построено с использованием архитектуры GraphQL, библиотека React для клиентской части в сочетании с Apollo Client для работы с GraphQL сервером. MS SQL использовался в качестве базы данных.

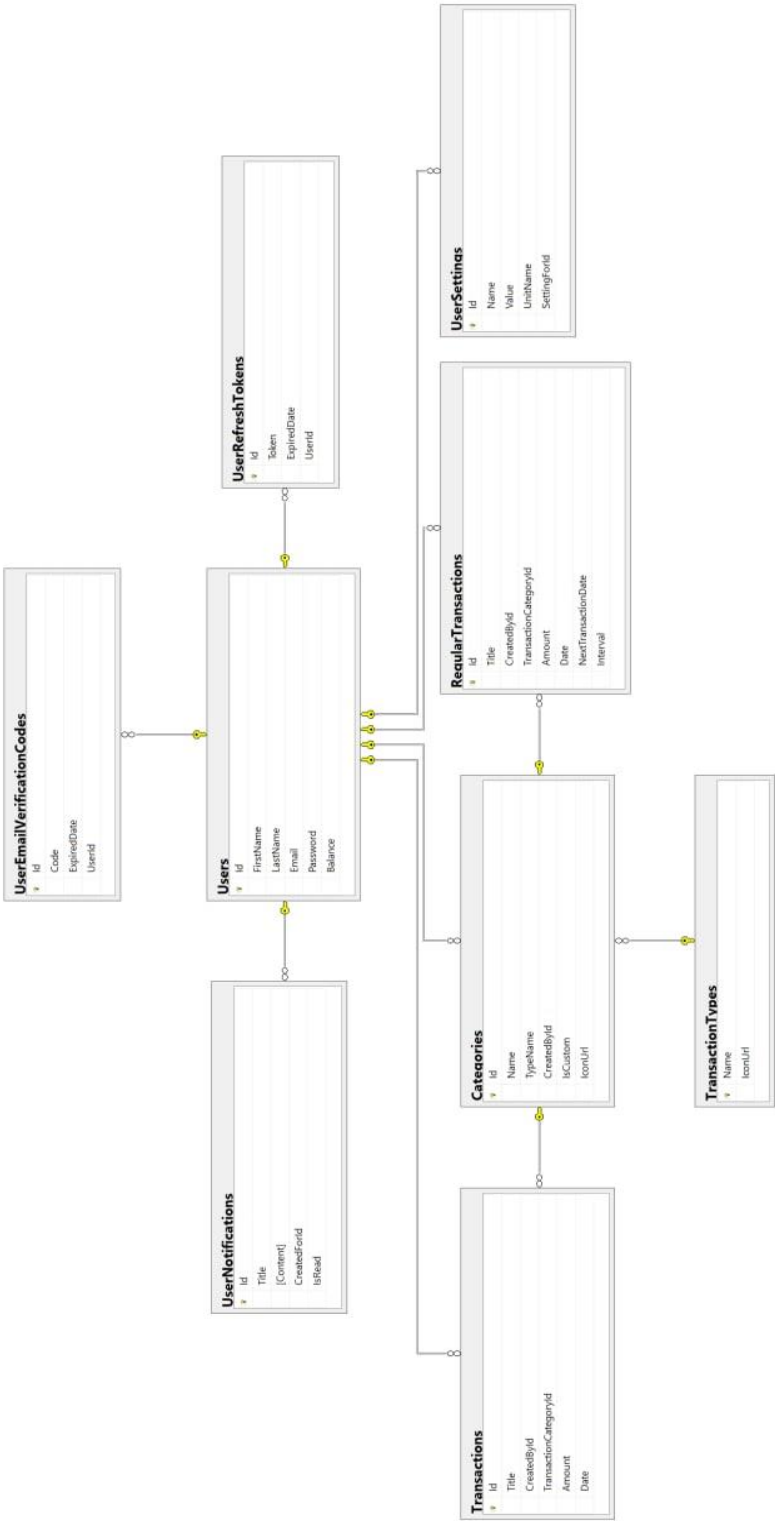
В соответствии с полученным результатом работы программы, можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список литературы

1. Руководство по ASP.NET Core 5 [Электронный ресурс]. — Режим доступа: <https://metanit.com/sharp/aspnet5/>. — Дата доступа: 13.11.2020.
2. Apollo GraphQL Client — разработка приложений на react.js без redux [Электронный ресурс]. — Режим доступа: <https://habr.com/ru/post/358292/>. — Дата доступа: 27.11.2020.
3. React Java Script library for building user interfaces [Электронный ресурс]. — Режим доступа: <https://reactjs.org/>. — Дата доступа: 27.11.2020.
4. Design Patterns - MVC Pattern [Электронный ресурс]. — Режим доступа: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm. — Дата доступа: 27.11.2020.
5. Введение в MS SQL Server и T-SQL [Электронный ресурс]. — Режим доступа: <https://metanit.com/sql/sqlserver/1.1.php>. — Дата доступа: 30.11.2020.
6. Docker [Электронный ресурс]. — Режим доступа: <https://docs.docker.com/>. — Дата доступа: 05.12.2020.

ПРИЛОЖЕНИЕ А

Логическая схема базы данных



ПРИЛОЖЕНИЕ Б

Файл используемых модулей package.json

```
{
  "name": "cash-scheduler-web-client",
  "version": "1.0.0",
  "author": {
    "name": "Ilya Matsuev",
    "email": "mirotvorec542546@gmail.com",
    "url": "https://github.com/IlyaMatsuev"
  },
  "engines": {
    "npm": "6.13.x",
    "node": "12.14.x"
  },
  "dependencies": {
    "@apollo/client": "^3.2.4",
    "@apollo/react-hooks": "^4.0.0",
    "chart.js": "^2.9.4",
    "graphql": "^15.3.0",
    "jwt-decode": "^3.0.0",
    "moment": "^2.29.1",
    "react": "^16.14.0",
    "react-chartjs-2": "^2.11.1",
    "react-dom": "^16.14.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^3.4.3",
    "react-semantic-toasts": "^0.6.5",
    "react-spring": "^8.0.27",
    "semantic-ui-calendar-react": "^0.15.3",
    "semantic-ui-css": "^2.4.1",
    "semantic-ui-react": "^2.0.0",
    "subscriptions-transport-ws": "^0.9.18"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "license": "MIT",
}
```

```
"browserslist": {  
  "production": [  
    ">0.2%",  
    "not dead",  
    "not op_mini all"  
  ],  
  "development": [  
    "last 1 chrome version",  
    "last 1 firefox version",  
    "last 1 safari version"  
  ]  
}
```

ПРИЛОЖЕНИЕ В

Методы, отвечающие за смену пароля пользователя

```

public async Task<string> CheckEmail(string email)
{
    var user =
ContextProvider.GetRepository<IUserRepository>().GetUserByEmail(email);

    if (user == null)
    {
        throw new CashSchedulerException("There is no
such user", new[] { nameof(email) });
    }

    string code = email.Code();
    var verificationCode = await
ContextProvider.GetRepository<IUserEmailVerificationCodeRepository>
().Update(
        new UserEmailVerificationCode(code,
DateTime.Now.AddMinutes(AuthOptions.EMAIL_VERIFICATION_CODE_LIFETIM
E), user)
    );

    var notificationDelegator = new
NotificationDelegator();
    var template = notificationDelegator.GetTemplate(
        NotificationTemplateType.VerificationCode,
        new Dictionary<string, string> { { "code",
verificationCode.Code } }
    );
    await Notificator.SendEmail(user.Email, template);
    await
ContextProvider.GetRepository<IUserNotificationRepository>().Create
(new UserNotification
    {
        Title = template.Subject,
        Content = template.Body,
        CreatedFor = user
    });

    return email;
}

public async Task<string> CheckCode(string email,
string code)
{

```

```

        var user =
ContextProvider.GetRepository<IUserRepository>().GetUserByEmail(email);

        if (user == null)
        {
            throw new CashSchedulerException("There is no
such user", new[] { nameof(email) });
        }

        var verificationCode =
ContextProvider.GetRepository<IUserEmailVerificationCodeRepository>
().GetByUserId(user.Id);
        if (verificationCode == null)
        {
            throw new CashSchedulerException("We haven't
sent you a code yet", new[] { nameof(email) });
        }

        if (verificationCode.ExpiredDate < DateTime.Now)
        {
            throw new CashSchedulerException("This code
has been expired", new[] { nameof(code) });
        }

        if (verificationCode.Code != code)
        {
            throw new CashSchedulerException("The code is
not valid", new[] { nameof(code) });
        }

        return await Task.FromResult(email);
    }

    public async Task<User> ResetPassword(string email,
string code, string password)
    {
        await CheckCode(email, code);

        if (string.IsNullOrEmpty(password))
        {
            throw new CashSchedulerException("Password is
a required field", new string[] { nameof(password) });
        }

        if (!Regex.IsMatch(password,
AuthOptions.PASSWORD_REGEX))
        {
            throw new CashSchedulerException(

```

```

        "Your password is too week. Consider to
        choose something with upper and lower case, digits and special
        characters with min and max length of 8 and 15",
        new string[] { nameof(password) }
    );
}

var user = await
ContextProvider.GetRepository<IUserRepository>().UpdatePassword(email, password);
var verificationCode =
ContextProvider.GetRepository<IUserEmailVerificationCodeRepository>
().GetByUserId(user.Id);
await
ContextProvider.GetRepository<IUserEmailVerificationCodeRepository>
().Delete(verificationCode.Id);
return user;
}

```